

PEDRO AMARAL





Definition: Network programmability is the use of software to deploy, manage, and troubleshoot network devices and services. This approach moves away from traditional manual configuration methods towards more automated, softwaredriven techniques.

#### **Allows**:

- Quicker deployments of new network configurations or services.
- More efficient managing of existing network infrastructure.
- Troubleshooting issues using programmatic tools

Instead of logging into each device individually to make changes, network programmability allows for centralized control and automation of these tasks.





#### **Key Components:**

**APIs** (Application Programming Interfaces):

- Sets of protocols, routines, and tools for building software applications. ullet
- Allow direct interaction with network devices.

Examples: Cisco's NX-API, Juniper's Junos XML API or REST APIs like **RESTCONF.** 

Protocols

**NETCONF** (Network Configuration Protocol): developed by the IETF for installing, manipulating, and deleting network device configurations.

Uses XML for data encoding and operates over a secure transport (typically SSH).

**RESTCONFA** REST-like protocol that provides a programmatic interface for accessing data defined in YANG, using datastores defined in NETCONF. Uses HTTP/HTTPS for transport and JSON or XML for data encoding.



Programming Languages:

**Python:** popular choice for network automation. Libraries like **Netmiko** or **NAPALM** are widely used for network automation tasks.

YANG: data modelling language used to model configuration and state data manipulated by network protocols like NETCONF and RESTCONF.

It describes what data is available, but not how it is stored or accessed.



### SHIFT IN APPROACH:

#### Traditional method: Manual, device-by-device configuration:

- Network engineers would log into each device separately.
- Use Command Line Interface (CLI) to make changes.
- Time-consuming and prone to human error, especially in large networks

#### New programmable method: Automated, programmatic control:

- Write scripts or use automation tools to manage multiple devices simultaneously.
- Changes can be version-controlled and tested before deployment.
- Enables consistency across the network and reduces the chance of human error.
- Allows for rapid deployment of changes and easier scaling of network management.







#### SHIFT IN APPROACH:

#### New programmable method: Automated, programmatic control:

- Write scripts or use automation tools to manage multiple devices simultaneously.
- Changes can be version-controlled and tested before deployment.
- Enables consistency across the network and reduces the chance of • human error.
- Allows for rapid deployment of changes and easier scaling of network management.

#### Examples:

- Configuration backups: Automatically backing up device configs daily.
- Compliance checks: Ensuring all devices meet security standards.
- Troubleshooting: Automated collection of diagnostic information.





### COMMON AUTOMATION TOOLS AND FRAMEWORKS

#### Network automation frameworks:

- Paramiko: Python library for connecting to network devices via SSH
- NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support): Provides a unified API to interact with different network device Operating Systems



#### Configuration management tools:

- Ansible: Agentless automation tool, uses YAML for playbooks
- Puppet: Uses its own declarative language, good for large-scale deployments
- Chef: Ruby-based, focuses on infrastructure as code



Still SSH based (manual CLI mimicking)



### **Network Programmability** KEY CONCEPTS AND BENEFITS

#### Automation

Examples:

- Configuration backups: Automatically backing up device configs daily.
- Compliance checks: Ensuring all devices meet security standards.
- Troubleshooting: Automated collection of diagnostic information.

- Defining the desired state of a network using simple text files
- IaC platforms for diferente purposes
- Vendor plugins specific for their network devices





#### laC (Infrastructure as Code)

### Treating network configurations as software code

Benifits:

• Version control, reproducibility, easier testing and rollback.

A	<mark>&gt;</mark> puppet 📀	NSO	
	Platforms		
	< 🍫 >		
, 56 L0.0.0/24''	SCM platform	H/Telnet/NETCONF/RESTC	ONF/etc

### COMMON AUTOMATION TOOLS AND FRAMEWORKS

#### Configuration management tools:

- Ansible: Agentless automation tool, uses YAML for playbooks
- Puppet: Uses its own declarative language, good for large-scale deployments
- Chef: Ruby-based, focuses on infrastructure as code

#### Network automation frameworks:

- Paramiko: Python library for connecting to network devices via SSH
- NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support): Provides a unified API to interact with different network device Operating Systems









Still SSH based (manula CLI mimicking)

### COMMON AUTOMATION TOOLS AND FRAMEWORKS

### APIs and Protocols

**NETCONF** (Network Configuration Protocol):

- RFC 6241
- Uses XML encoding
- Provides a set of operations to manage device configurations
- Runs over SSH, port 830 as default.
- Yang Models are used to model the device's config.







Content Configuration/Notification data

#### Operations

Messages <rpc> <rpc-reply> <notification>

> Secure Transport SSH, TLS



CONFIGURATION AS CODE PRINCIPLES

- Defining network configurations in a declarative manner (models, automation scripts)
- Version control for network configurations (for example using git for change tracking and rollbacks)
- Treating network changes like software development (code reviews, testing)





### er (models, automation scripts) le using git for change tracking and rollbacks) code reviews, testing)

EXAMPLE: AUTOMATING VLAN CONFIGURATION WITH ANSIBLE

Scenario: Configuring VLANs across multiple switches using Ansible to orchestrate and automate CLI configuration.

Step-by-step walkthrough:

• a. Define the desired VLAN configuration in YAML (vlan\_config.yml):

vlans: - id: 10 name: "Data" - id: 20 name: "Voice" - id: 30 name: "Guest"



#### EXAMPLE: AUTOMATING VLAN CONFIGURATION WITH ANSIBLE

Step-by-step walkthrough:

b. Create an Ansible playbook to apply the VLAN configuration (configure\_vlans.yml):

Ansible translates the configuration specified in the cisco.ios.ios\_vlans module a into the corresponding CLI commands, which it then sends to the device over SSH.

• c. Run the playbook and verify the results :

ansible-playbook -i inventory.yml configure\_vlans.yml



```
- name: Configure VLANs on switches
 hosts: switches
 gather_facts: no
  vars_files:
    - vlan_config.yml
  tasks:
   - name: Configure VLANs
      cisco.ios.ios_vlans:
        config:
          - name: "{{ item.name }}"
            vlan_id: "{{ item.id }}"
            state: merged
      loop: "{{ vlans }}"
    - name: Save running config to startup config
      cisco.ios.ios_config:
        save_when: always
    - name: Verify VLAN configuration
      cisco.ios.ios_command:
        commands:
          - show vlan brief
      register: vlan_output
    - name: Display VLAN configuration
      debug:
        var: vlan_output.stdout_lines
```

### EXAMPLE: AUTOMATING VLAN CONFIGURATION WITH ANSIBLE

Step-by-step walkthrough:

• b. Run the playbook and verify the results ): ans

inventory.yml file list all the hosts that we want to configure:



ansible-playbook -i inventory.yml configure\_vlans.yml

<pre>switches:</pre>
hosts:
switch1:
ansible_host: 192.168.1.10
ansible_user: admin
ansible_password: your_password
ansible_network_os: ios
switch2:
ansible_host: 192.168.1.11
ansible_user: admin
ansible_password: your_password
ansible_network_os: ios

EXAMPLE: AUTOMATING VLAN CONFIGURATION WITH ANSIBLE

Benefits:

- Consistency across devices.  $\bullet$
- Easy to modify and extend. ullet
- Version controlled.  $\bullet$
- Idempotent (can be run multiple times safely). ullet
- Includes verification step.  $\bullet$
- Separates data (VLAN definitions) from logic (automation steps). ullet



### RESTCONF

**RESTCONF** is an API :

- Used instead of CLI.
- Orchestration tools like Ansible can use RESTCONF modules and orchestrate configurations via RESTCONF and other APIs.

**RESTCONF vs NETCONF:** 

- NETCONF uses XML for data encoding and typically runs over SSH.
- RESTCONF uses HTTP/HTTPS and supports both XML and JSON. lacksquare
- NETCONF provides a more extensive set of operations but can be more complex to lacksquareimplement.
- RESTCONF aligns more closely with RESTful design principles. •

#### Key benefits: simplicity, HTTP-based, JSON support



RESTCONF ARCHITECTURE

REST API :

- HTTPS-based communications.
- Stateless
- RESTful API interface for operations
- Verbs for CRUD actions
- Standarized response codes







### **Network Programmability** RESTCONF ARCHITECTURE

RESTCONF protocol:

- RFC 8040.
- Based on XML or JSON for data encoding
- Using YANG data models





Content (Config/Operational Data) XML, JSON

Operations

GET, POST, PUT, PATCH, DELETE

Transport HTTPS



### RESTCONF ARCHITECTURE

YANG data models

- RFC 6020.
- Data modeling language
- Models configurations and state of a data of a device or service
- Organized in nodes
- Several node and data types
- Device Data Models (Interface, VLAN, etc)
- Service Data Models (L3VPN, VRF, etc)
- Industry Santard vs. Vendor Specific





#### YANG DATA MODELS





Sequence of leaf nodes with exactly one value of a particular type per leaf



description "List of domain names to search";

#### YANG DATA MODELS





Sequence of entries uniquely identified by the value on their key leaf. A list can define multiple keys and may contain any number of child nodes of any type



```
"Message given at start of login session";
Grouping of related nodes in a subtree. It has only child nodes and no value.
          May contain any number of child nodes of any type
            tailf:info "Assign interface as port0";
```

```
tailf:info "Interface Name";
```

### RESTCONF EXAMPLE

- 1. Creating a new VLAN (POST request)
- Import the necessary libraries (requests and json).
- Disable SSL warnings for sandbox enviroments.
- Define the base URL for RESTCONF on Cisco NX-OS (sandbox environment)
- Set headers to indicate the content type and format (YANG)
- Provide authentication details

im im # re # ur he

> } au



```
import requests
import json
```

# Disable SSL verification warnings (for sandbox environment)
requests.packages.urllib3.disable\_warnings()

```
# DevNet Always-On NX-OS Sandbox details
url_base = "https://sandbox-nxos-1.cisco.com/restconf"
headers = {
    "Accept": "application/yang-data+json",
    "Content-Type": "application/yang-data+json"
}
```

auth = ("admin", "Admin\_1234!")



### RESTCONF EXAMPLE

2. Retrieving interface information (GET request)

- Construct the URL to get interface information
- Send GET request to retrieve interface data.
- Check the response status and print the interfaces in a readable format if successful



```
# Get all interfaces
url = f"{url_base}/data/Cisco-NX-OS-device:System/intf-items"
response = requests.get(url, headers=headers, auth=auth, verify=False)
```

```
if response.status_code == 200:
    interfaces = response.json()
    print(json.dumps(interfaces, indent=2))
else:
    print(f"Error: {response.status_code} - {response.text}")
```



**RESTCONF EXAMPLE** 

- **Container** (System): The top-level container for system-level  $\bullet$ configurations.
- Contains all interface-related (intf-items): Container  $\bullet$ configurations.
- **Container** (phys-items): Specifically for physical interfaces **List** (PhysIf): List of physical interfaces. Each interface is an entry ullet
- in the list.
  - Leaf id: Identifier for the interface.
  - Leaf descr: Description of the interface.
  - **Leaf** adminSt: Administrative state of the interface (e.g., up or d own)



#### Interface YANG Model Example



### RESTCONF EXAMPLE

3. Modifying interface description (PATCH request)

- Define the Interface to be updated
- Create a payload with the new description
- Send a PATCH request to update the interface description
- Check the response status confirm the update

```
# Update description for Ethernet1/1
interface_name = "eth1/1"
url = f"{url_base}/data/Cisco-NX-OS-device:System/intf-items/phys-items/PhysIf={interface_name}"
payload = {
    "Cisco-NX-OS-device:PhysIf": {
        "descr": "Updated via RESTCONF"
}
response = requests.patch(url, headers=headers, auth=auth, data=json.dumps(payload), verify=False)
if response.status_code == 204:
    print(f"Interface {interface_name} description updated successfully")
else:
    print(f"Error: {response.status_code} - {response.text}")
```





### RESTCONF EXAMPLE

4. Creating a new VLAN (POST request)

- Construct the URL to create a new VLAN
- Create a payload with the VLAN details
- Send a POST request to create the VLAN
- Check the response status and confirm the creation



else:



```
# Create VLAN 100
url = f"{url_base}/data/Cisco-NX-OS-device:System/bd-items/bd-items"
    "Cisco-NX-OS-device:bd-items": {
        "BD-list": [
                "fabEncap": "vlan-100",
                "name": "VLAN100_via_RESTCONF"
```

response = requests.post(url, headers=headers, auth=auth, data=json.dumps(payload), verify=False)

```
if response.status_code == 201:
   print("VLAN 100 created successfully")
```

print(f"Error: {response.status\_code} - {response.text}")



### RESTCONF EXAMPLE

VLAN YANG Model Example

- **Container** (System): The top-level container for system-level configurations.
- **Container** (bd-items): Contains all bridge domain (VLAN) related • configurations.
- List (PhysIf): List of bridge domains (VLANs). Each VLAN is an entry in the list.
  - **Leaf** fabEncap: Encapsulation method, including the VLANN ID.
  - Leaf name: Name of the VLAN.



```
"Cisco-NX-OS-device:System": {
  "bd-items": {
    "BD-list": [
      {
        "fabEncap": "vlan-100",
        "name": "VLAN100_via_RESTCONF"
      }
```



### RESTCONF EXAMPLE

4. Retrieving routing information (GET request)

- Construct the URL to get IPv4 routing information
- Send a GET request to retrieve routing data
- Send a POST request to create the VLAN
- Check the response status and print the routes in a readable format if successful

```
# Get IPv4 routing table
url = f"{url_base}/data/Cisco-NX-OS-device:System/ipv4-items/inst-items/dom-items/Dom-list=default/rt-items"
response = requests.get(url, headers=headers, auth=auth, verify=False)
if response.status_code == 200:
    routes = response_json()
   print(json.dumps(routes, indent=2))
else:
    print(f"Error: {response.status_code} - {response.text}")
Explanation:
Construct the URL to get IPv4 routing information.
Send a GET request to retrieve routing data.
Check the response status and print the routes in a readable format if successful.
```





### **RESTCONF EXAMPLE**

VLAN YANG Model Example

- **Container** (System): The top-level container for system-level configurations.
- **Container** (ipv4-items): Contains all IPv4-related configurations.
- **Container** (inst-items): Instance-level configurations for IPv4.
- **Container** (dom-items): Domain-related configurations.
- List (Dom-list): List of domains. Each domain is an entry in the list.
  - Leaf (name): Name of the domain (e.g., default).
- **Container** (rt-items): Contains routing table entries
- **List** (Route-list): List of routes. Each route is an entry in the list.
  - Leaf (prefix): Prefix of the route.
  - Leaf (nextHop): Next-hop address for the route. ullet



```
"Cisco-NX-OS-device:System": {
  "ipv4-items": {
    "inst-items": {
      "dom-items": {
        "Dom-list": [
            "name": "default",
            "rt-items": {
              "Route-list":
                  "prefix": "10.0.0/24",
                  "nextHop": "192.168.0.1"
```

